IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Rajalakshmi Ramasubramanian
21 October 2025

Outline

Executive Summary

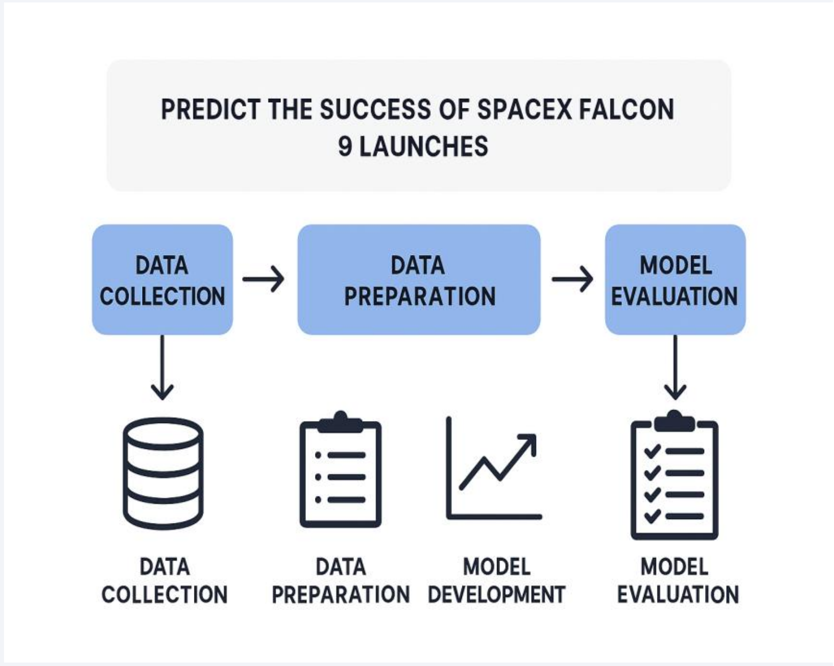
- Summary of methodologies
- Summary of all results

Image created using Perplexity AI

Introduction

In a galaxy far, far away…..a data scientist wanted to understand how SpaceX was able to run its Falcon 9 rocket launch programme at less than half the cost of its' peers (62 Mn vs approx. 165 Mn). She understood that the trick was in reusing the first stage of the launch ; hence, predicting a successful launch would help her build her dream space company that could rival SpaceX one day.

Luckily for her, she had learnt the tools of the trade from the IBM Data Science Certification programme. She took this challenge as a part of her Capstone Project and used different cool (and nerdy) data science techniques to predict the success of each launch considering the different factors that could influence this. "No mumbo jumbo here, just plain numbers", she muttered, as she jotted various codes and tested them out using the data available for the launches.



Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:

  Gather features: launch longitude/latitude, payload mass and orbit, launch date, launch site, and outcome (success/failure) from SpaceX REST API and Web scraping from Wikipedia for Falcon 9 launches

- Perform data wrangling

  Clean and preprocess the dataset by removing irrelevant entries, handling missing values, encoding categorical features using One Hot Encoding, and adding a target column for landing success.

- Perform exploratory data analysis (EDA) using visualization and SQL

  Visualize key variables and trends influencing launch success—such as payload mass, site, and flight count—via plots and correlation analysis.

Methodology

Executive Summary

- Perform interactive visual analytics using Folium and Plotly Dash

  Build geographic launch maps using Folium

  Create dashboards and plot trends using Plotly Dash

- Perform predictive analysis using classification models

  Build and compare classification models (Decision Tree, SVM, KNN, Logistic Regression) to predict launch success, optimizing their performance.
  Evaluate models using test data and cross-validation, reporting accuracy and other metrics.

Data Collection

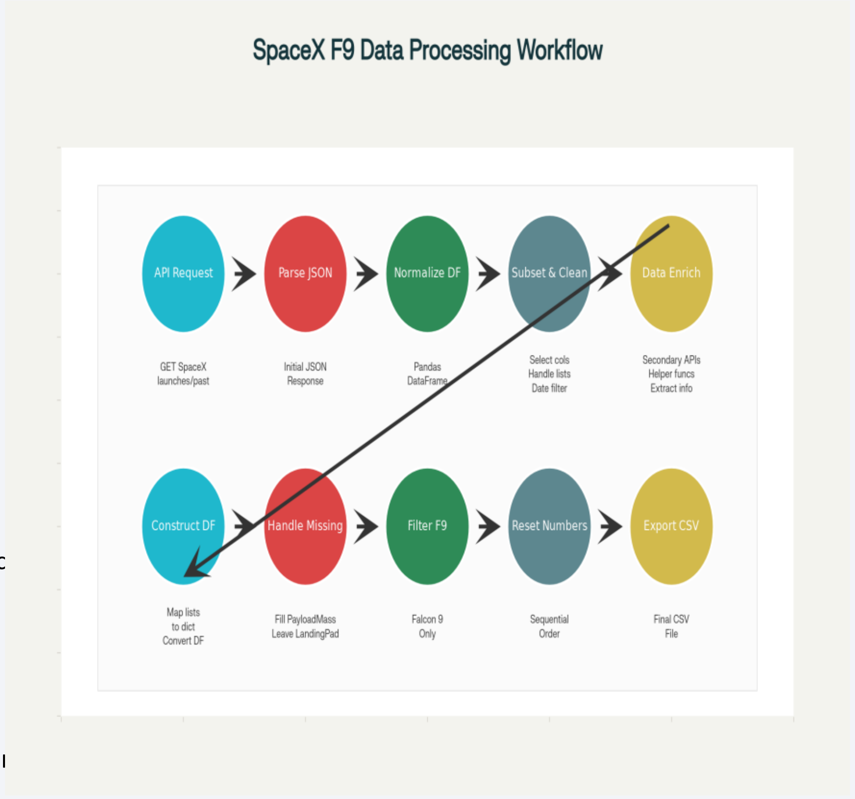
The process combines web scraping for a baseline historical overview and API calls for more detailed, structured, and potentially up-to-date information.

- **API Calls (SpaceX REST API):** Retrieves detailed, structured data directly from the SpaceX API. This involves fetching a list of past launches and then making subsequent calls using IDs obtained from the initial list to gather specific details about rockets, launchpads, payloads, and core stages.

- **Web Scraping (Wikipedia):** Extracts launch records from a static snapshot of the "List of Falcon 9 and Falcon Heavy launches" Wikipedia page. This provides a foundational dataset with key launch details as presented in the wiki tables.

Data Collection – SpaceX API

- **Initial API Request:** Make an HTTP GET request to the main SpaceX API endpoint for past launches (https://api.spacexdata.com/v4/launches/past) using the requests library. For consistency in the lab, a static JSON response URL is used instead of hitting the live API directly.
- **Parse Initial JSON:** Decode the JSON response into a Python object (list of dictionaries).
- **Normalize to DataFrame:** Use pandas.json_normalize() to convert the complex JSON structure into a flat Pandas DataFrame (df).
- **Subset and Initial Cleaning:**
  - Select key columns containing IDs and essential info (rocket, payloads, launchpad, cores, flight_number, date_utc).
  - Handle potential variations in cores and payloads structure (ensure they are lists and extract the first element, assuming single core/payload focus for simplicity in this lab).
  - Convert the date_utc string to a datetime object and extract only the date part.
  - Filter the DataFrame to include only launches up to a specific date (November 13, 2020) for consistency.
- **Data Enrichment via Secondary API Calls:**
  - Initialize empty lists to hold the detailed data to be fetched (e.g., BoosterVersion, PayloadMass, Orbit, LaunchSite, Longitude, Latitude, Outcome, Flights, GridFins, Reused, Legs, LandingPad, Block, ReusedCount, Serial). Define and use helper functions (getBoosterVersion, getLaunchSite, getPayloadData, getCoreData).
  - Each helper function iterates through the relevant column (e.g., rocket, launchpad, payloads, cores) in the initially created DataFrame. Inside the functions, extract the ID (e.g., rocket ID, launchpad ID). Make *additional API calls* using these IDs to specific endpoints (e.g., https://api.spacexdata.com/v4/rockets/{id}, https://api.spacexdata.com/v4/payloads/{id}, etc.).
  - Parse the JSON response from these secondary calls. Extract the required detailed information (e.g., rocket name, payload mass, launchpad name/coordinates, core reuse count, landing success/type).
  - Append the extracted details to the corresponding global lists initialized earlier.
- **Construct Final DataFrame:** Create a new dictionary (launch_dict) mapping descriptive column names to the populated lists. Conver this dictionary into the final Pandas DataFrame (launch_data).
- **Handle Missing Data:** Calculate the mean PayloadMass and fill any missing (NaN) values in that column with the mean. (LandingPad NaNs are kept as they indicate no landing pad was used/targeted).
- **Filter for Falcon 9:** Remove entries corresponding to 'Falcon 1' launches, keeping only 'Falcon 9' data. Store this in data_falcon9.
- **Reset Flight Numbers:** Renumber the FlightNumber column sequentially for the filtered Falcon 9 dataset.
- **Export:** Save the final, cleaned, and enriched Falcon 9 DataFrame to a CSV file (dataset_part_1.csv).



Git Link to the code: Link.

Data Collection - Scraping

1.Fetch HTML: Use the requests library to send an HTTP GET request to the specified static Wikipedia URL (an older version to ensure consistency).

2.Parse HTML: Create a BeautifulSoup object from the fetched HTML content, allowing for easy navigation and searching of the HTML structure.

3.Identify Target Tables: Locate the specific HTML tables (<table>) that contain the launch records (identified by class wikitable plainrowheaders collapsible in the code).

4.Extract Headers: Find all header elements (<th>) within the target table(s) and use a helper function (extract_column_from_header) to clean and extract the column names.

5.Initialize Data Structure: Create an empty Python dictionary (launch_dict) where keys are the extracted column names, and values are initially empty lists. Add specific keys for 'Date' and 'Time' extracted separately later.

6.Iterate Through Rows: Loop through each row (<tr>) in the identified table(s).

7.Identify Launch Data Rows: For each row, check if it represents a launch entry (e.g., by verifying if the first header cell <th> contains a digit representing the flight number).

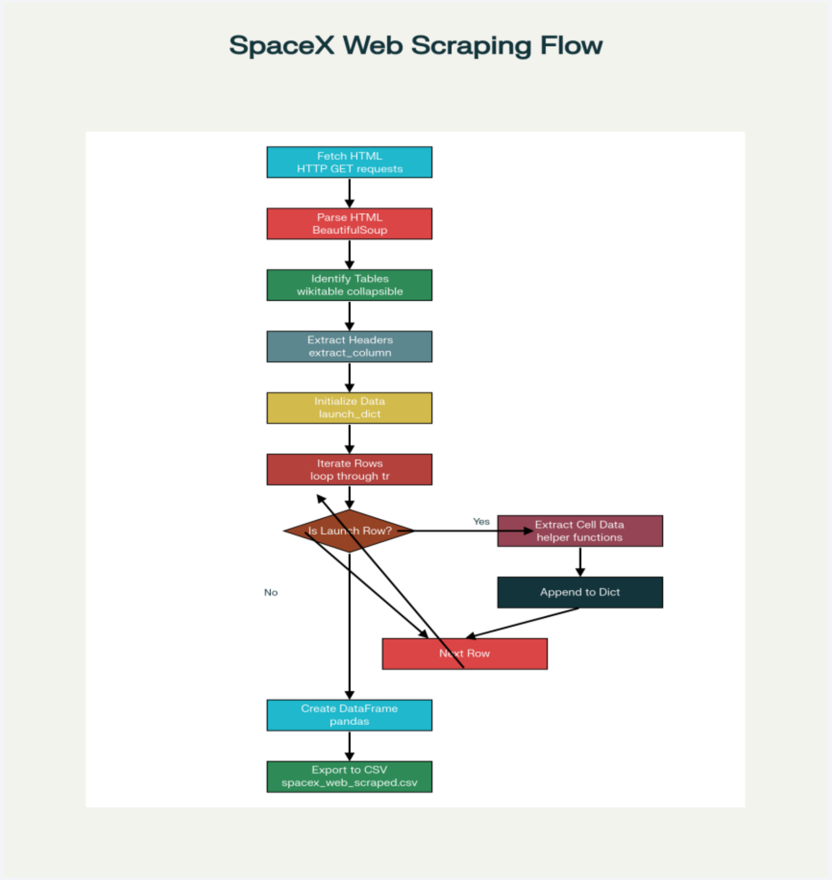
8.Extract Cell Data: If it's a launch row, find all data cells (<td>) within that row.

9.Parse and Clean Data: Use custom helper functions (date_time, booster_version, landing_status, get_mass) and direct BeautifulSoup object manipulation (e.g., row[2].a.string for launch site) to extract and clean the text content from each relevant cell. This involves handling different formats, removing annotations (like [b]), and extracting specific parts (like date vs. time).

10.Populate Dictionary: Append the extracted, cleaned data points (Flight Number, Date, Time, Booster Version, Launch Site, Payload, Payload Mass, Orbit, Customer, Launch Outcome, Booster Landing) to their respective lists within the launch_dict.

11.Create DataFrame: Convert the populated launch_dict into a Pandas DataFrame.

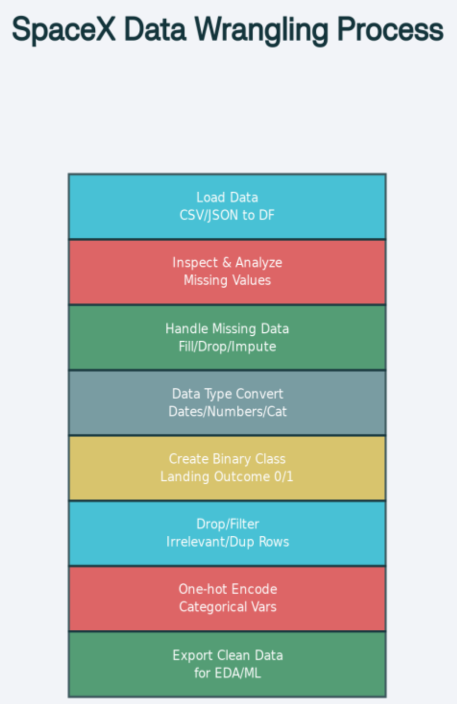
12.Export: Save the DataFrame to a CSV file (spacex_web_scraped.csv).



Git Link to the code: Link.

Data Wrangling

1. Import Libraries
•import pandas as pd and import numpy as np
2. Load, inspect and analyse data
•Quick check: df.head(), Check types: df.dtypes, Find missing: df.isnull().sum(), check for value_counts to check for number and occurrence of each orbit and outcome
3. Handle Missing Values
•Fill missing payload mass: df['PayloadMass'].fillna(df['PayloadMass'].mean(), inplace=True)
•Retain NaNs in LandingPad as in dicator
4. Filter Useful Launch Data
•Keep Falcon 9 launches: df[df['BoosterVersion'] == 'Falcon 9’]
5. Convert Data Types
•Convert to datetime
6. Encode Landing Outcome to Binary
•Transform text outcomes to numeric: df['Class'] = [0 if outcome in badoutcomes else 1 for outcome in df['Outcome’]]
7. Calculate the average success rate


SpaceX Data Wrangling Process
Load Data CSV/JSON to DF
Inspect & Analyze Missing Values
Handle Missing Data Fill/Drop/Impute
Data Type Convert Dates/Numbers/Cat
Create Binary Class Landing Outcome 0/1
Drop/Filter Irrelevant/Dup Rows
One-hot Encode Categorical Vars
Export Clean Data for EDA/ML

Git Link to the code: Link.

EDA with Data Visualization

1. Summary of Charts Plotted and Their Purpose

•Bar Charts:

Used to visualize the distribution of launches by site, orbit type, and year. This allows us to easily compare the success rates for each orbit for example(refer to the picture)

Purpose: Identify busiest launch sites, most common orbits, and yearly launch trends.

•Scatter Plots:

Payload mass vs landing outcome, payload mass trends over the years., , visualize number of flights for each orbit (refer to the chart)

Purpose: Analyze relationship between key variablesetc.

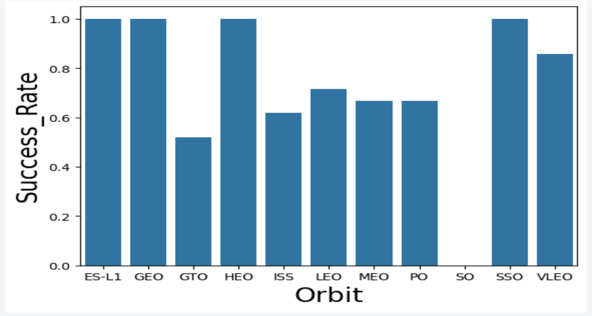
•Line Charts

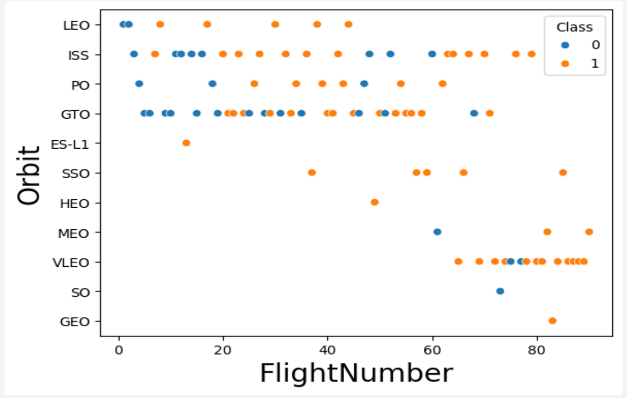
Allows to see the key trend over time for success rates etc and to infer if these have been increasing over time

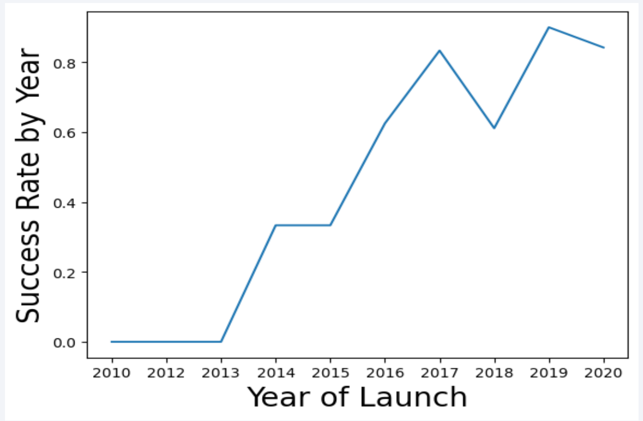
Why these charts are used:

•To clearly show trends, relationships, geographic insights, and outlier points.

•To connect features (payload, site, orbit, year, customer) to success/failure outcomes







Git Link to the code: Link.

EDA with SQL

- Display the names of the unique launch sites in the space mission
- Get 5 rows where launch sites begin with KSC string
- Display the total payload mass carried by boosters launched by NASA (CRS)
- Display average payload mass carried by booster version F9 v1.1
- List the date where the successful landing outcome in drone ship was achieved.
- List the names of the boosters which have success in ground pad and have payload mass greater than 4000 but less than 6000
- List the total number of successful and failure mission outcomes
- List all the booster_versions that have carried the maximum payload mass using a subquery
- List the records which will display the month names, succesful landing_outcomes in ground pad ,booster versions, launch_site for the months in year 2017
- Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order

Git Link to the code: Link.

Build an Interactive Map with Folium

- 1. Map Objects Added:
- Markers
  Points added to indicate each SpaceX launch site or specific launches (often with custom colors for success vs. failure).
- Marker Clusters
  Grouped markers to visually declutter and show clusters of successful and failed launches at each site.
- Circles
  Plotted to emphasize each site’s location, often with radius proportional to significance or for zoomed-out views.
- Lines
  Drawn between launch sites and other locations (e.g., proximity analysis, showing distances).
- Pop-up Labels
  Display additional information (site name, launch outcome, payload) when clicking on markers or circles.
- 2. Why These Objects Were Added:
- Markers: Used to pinpoint the exact coordinates for launch sites and missions.
- Marker Clusters: Helped depict multiple launches at a single location, differentiating success/failure.
- Circles: Provided clear visual emphasis and grouping for launch locations on zoomed-out maps.
- Lines: Illustrated important geospatial relationships (e.g., distance to nearest city or other sites).
- Pop-up Labels: Enabled instant access to relevant metadata for peer reviewers and viewers, improving map interactivity.

Git Link to the code: Link

Build a Dashboard with Plotly Dash

Following details were added to the dashboard using Plotly Dash

a) Drop-down to list the launch sites with an option to select “All Sites”

b) Pie Chart showing the number of launches overall and success and fail percentages per launch site

c) Range slider to allow for a range of payload mass to be considered (minimum and maximum values set)

d) Scatter plot to show the success/outcome per payload

Interactive charts allow users to further investigate (such as looking at outcome per launch site while also assessing impact of payload on outcome). It is also easier for users to navigate without having technical knowhow to code each time the selection criteria to filter for launch site or payload mass.

Git link to Code: Link

Predictive Analysis (Classification)

1. Data Load and transformation: Data was loaded and scaling transformation was applied using StandardScaler() .
2. Train-test Split: Data was then split into train and test samples.

3. Four models were chosen for the purpose of testing - Logistic Regression, Support Vector Machine (SVM), Decision Tree, and K-Nearest Neighbors (KNN). This was done using the Scikit learn package and models were optimised using GridSearchCV through cross validation for 10 folds.

Employed GridSearchCV for hyperparameter tuning:

- Logistic Regression → optimized C (regularization).
- KNN → optimized n_neighbors.
- SVM → optimized kernel and C.
- Decision Tree → optimized max_depth and criterion.

Selected best estimator from GridSearchCV results (highest validation accuracy).

3. Model Evaluation:

•Performance metrics used: Accuracy, Confusion Matrix, and Cross-Validation Scores.
•Evaluated models using .score() and confusion_matrix() from sklearn.metrics.
•Compared test accuracy of each model to identify consistency and generalization.

4. Best Performing Model: Chose the best performing model based on the results noted across both testing and training data

Git link to Code: Link

Results

Exploratory data analysis results

- There are 4 unique launch sites in the database and of the total launches, 61 are successes and 10 are failures.
- The launch success rate has significantly increased over time, particularly stabilizing at a high rate in the later years of the dataset.
- As the flight number increases, the first stage is more likely to land successfully. The payload mass also appears to be a factor; even with more massive payloads, the first stage often returns successfully.
- More launches are noted against the site CCAFS SLC-40. For the LEO orbit, success seems to be related to the number of flights. Conversely, in the GTO orbit, there appears to be no relationship between flight number and success. With heavy payloads the successful landing or positive landing rate are more for Polar, LEO and ISS. However, for GTO, it's difficult to distinguish between successful and unsuccessful landings as both outcomes are present.

- **Interactive analytics demo in screenshots**
  - From the Folium map - All launch sites were visually confirmed to be located near the coast and relatively close to the equator (providing a velocity advantage). They are also within reach of cities, railways and roads allowing for easy logistic management.
  - For the interactive dashboard - We can see that KSC LC-39A had the most successful launches and this achieved a 76.9% success rate.
  - Lower payloads seem to have higher success outcomes.

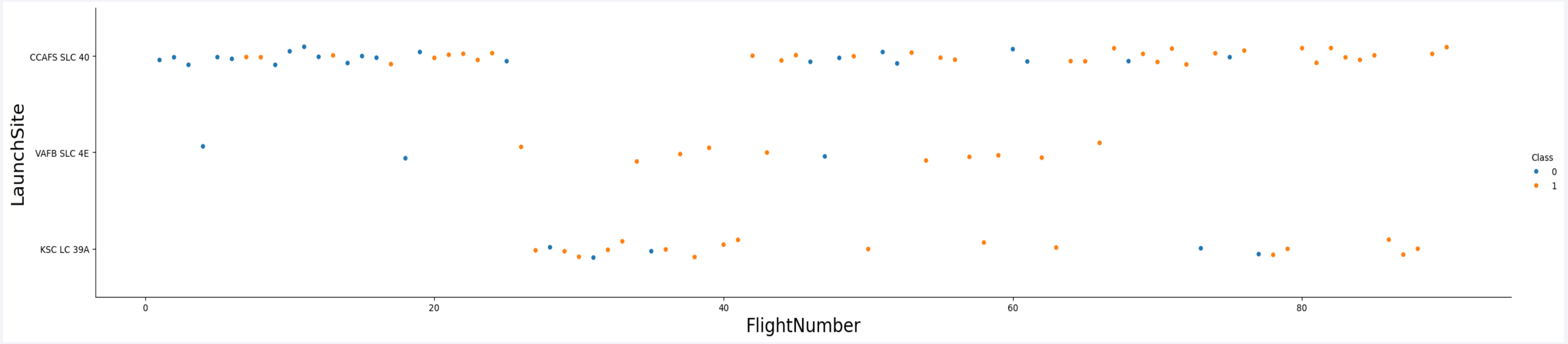
- **Predictive analysis results**
  - Logistic Regression, SVM and KNN are the best models. Decision tree approach underperforms on the testing set indicating overfitting to the training data.

Section 2

Insights drawn from EDA

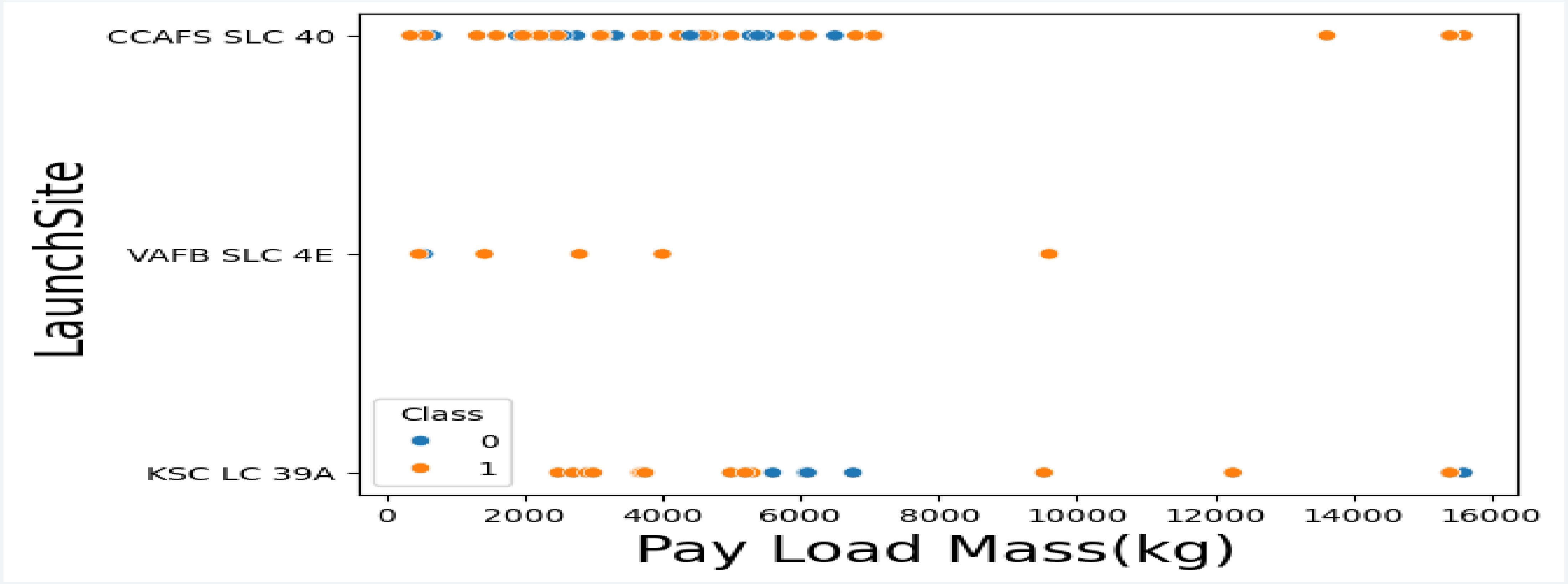
Flight Number vs. Launch Site

- Show a scatter plot of Flight Number vs. Launch Site



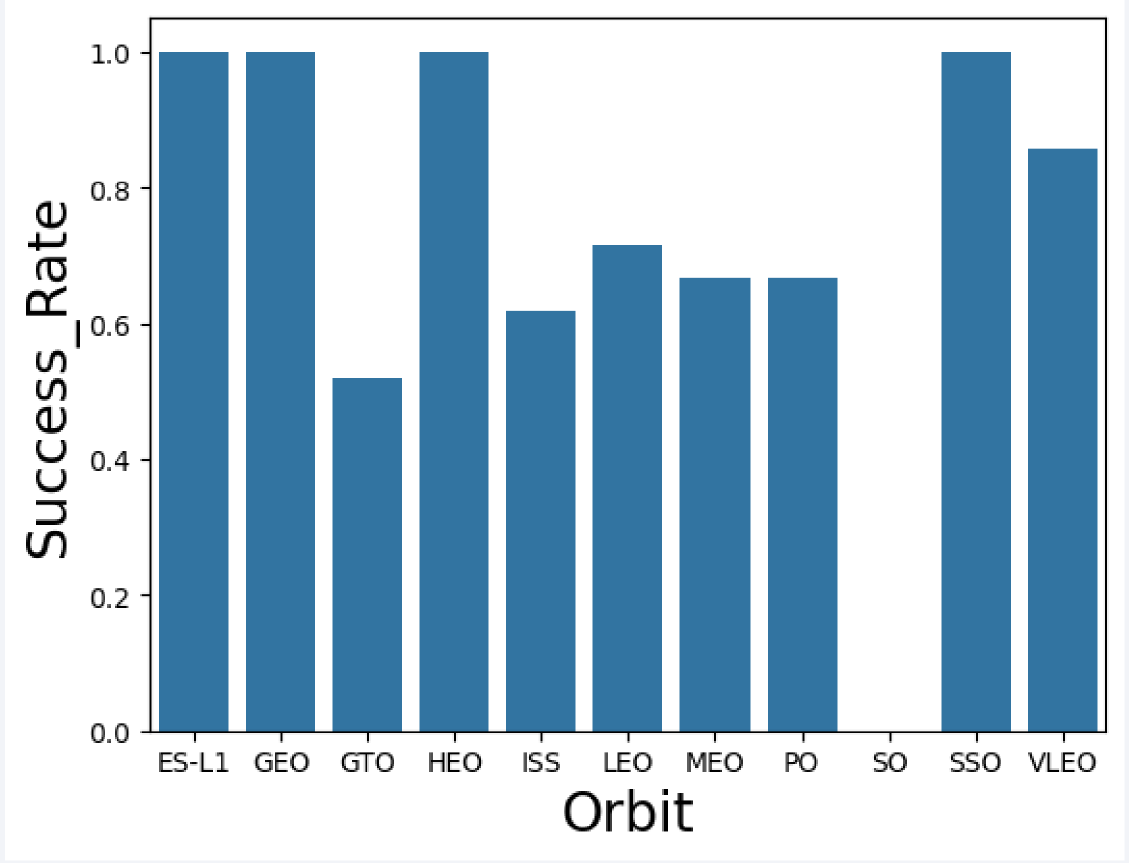
The highest number of launches is from CCAFS SLC-40. The higher the number of flights, the better the outcome.

Payload vs. Launch Site



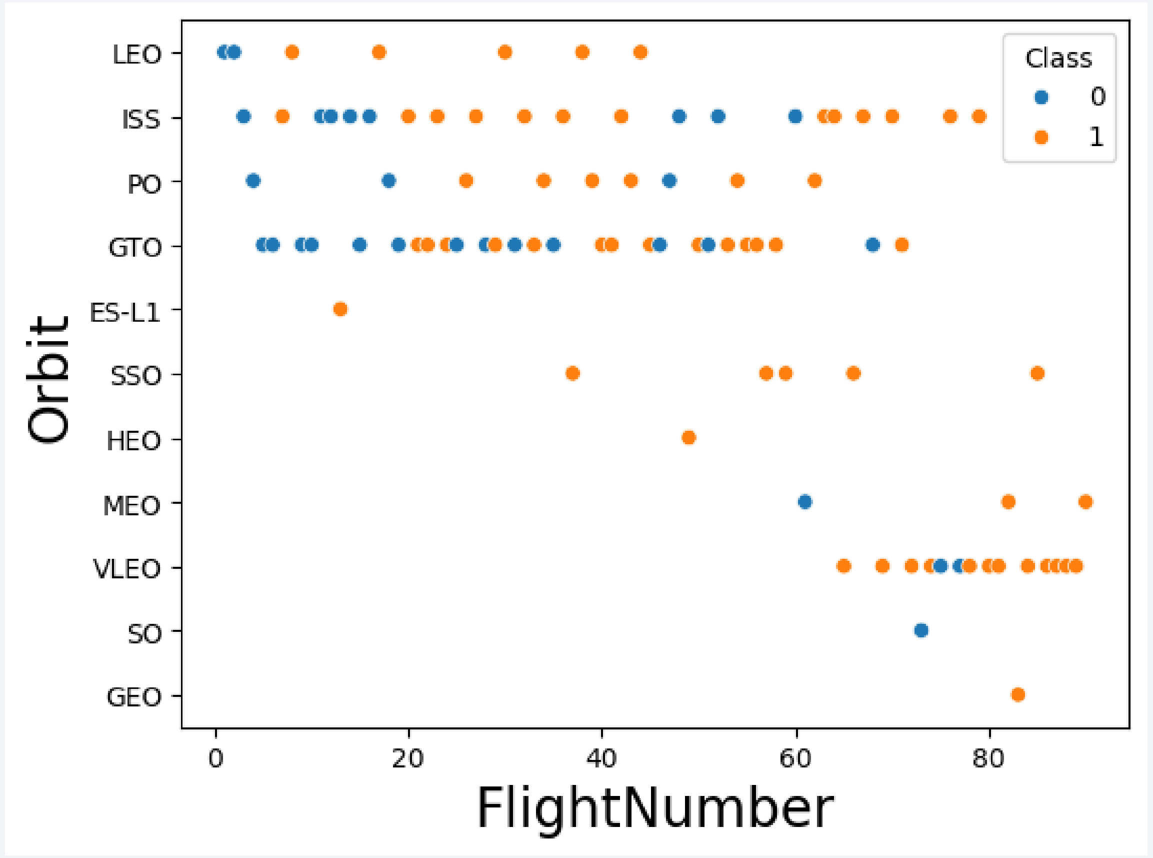
Most launches have had lower payloads (<7000). The site CCAGS SLC40 has seen successes for very heavy payloads. Statistical significance for high payload success rates has not been established. The site VAFB SLC 4E has mainly dealt with lower payloads.

Success Rate vs. Orbit Type



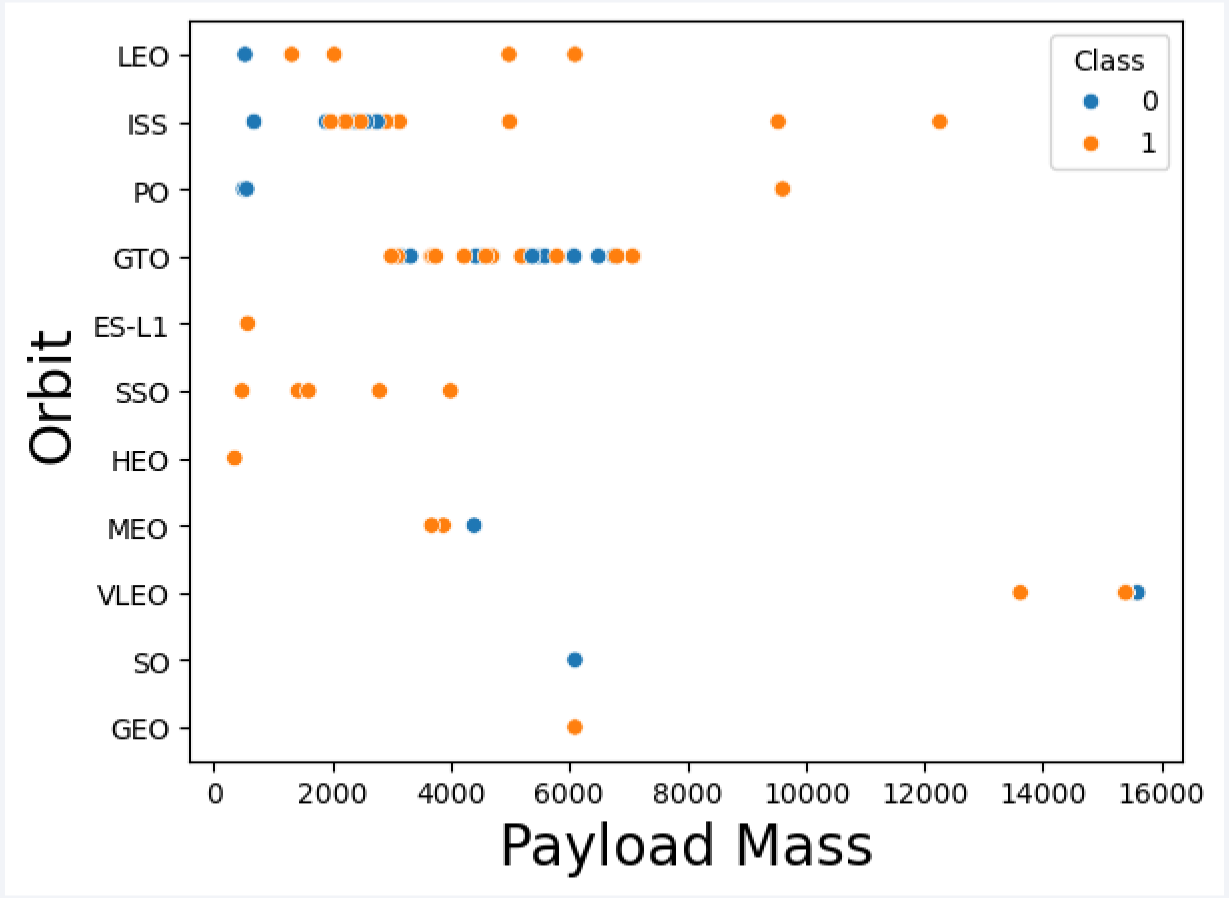
Success rates are the least for SO, GTO and ISS orbits. GEO, ES-L1, SSO and HEO orbits have seen perfect launches (denoted by a success rate of 100%)

Flight Number vs. Orbit Type



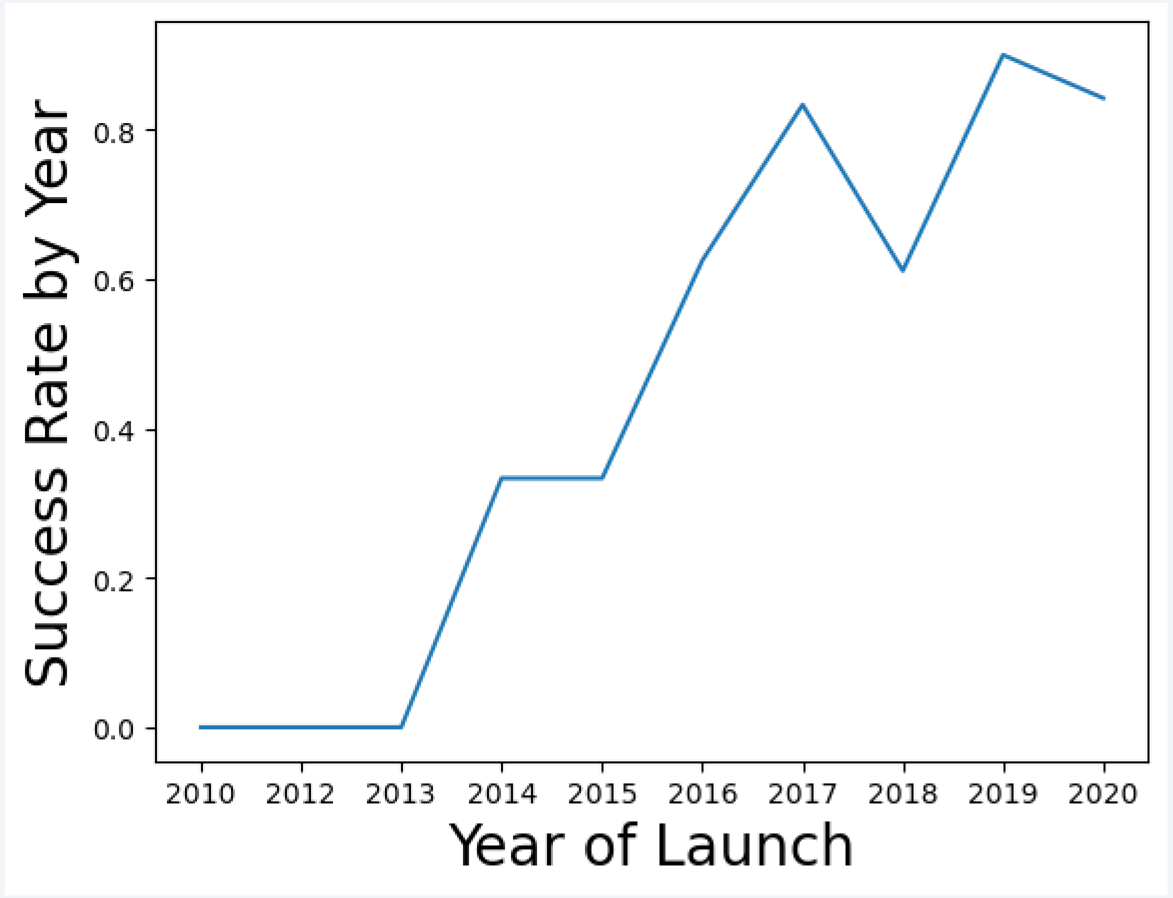
In the LEO orbit, success seems to be related to the number of flights. Conversely, in the GTO orbit, there appears to be no relationship between flight number and success. There are no successful launches for the SO orbit.

Payload vs. Orbit Type



With heavy payloads the successful landing or positive landing rate are more for Polar,LEO and ISS.
However, for GTO, it's difficult to distinguish between successful and unsuccessful landings as both outcomes are present.

Launch Success Yearly Trend



Success rates have been increasing since 2013 on a y-o-y basis showing that experience has helped improve the overall process

All Launch Site Names

```
# Changing the code to allow for connection via SQlite considering
compatibility issues
conn = sqlite3.connect('my_data1.db')

# Create a cursor object
cur = conn.cursor()
# Execute the SQL query to get 5 rows where launch sites begin with KSC string
cur.execute('SELECT * FROM SPACEXTABLE WHERE UPPER("LAUNCH_SITE")
LIKE "KSC%" LIMIT 5')

# Fetch all results
results = cur.fetchall()

# Print the results
for row in results:
    print(row)

# Close the connection
conn.close()
```

‘Distinct’ is used to identify unique launch_sites from the database. The cursor is used as this is a SQL conn query. There are 4 unique launch sites noted from the data

Result of the query:

```
('CCAFS LC-40',)
('VAFB SLC-4E',)
('KSC LC-39A',)
('CCAFS SLC-40',)
```

Launch Site Names Begin with 'KSC'

```
# Changing the code to allow for connection via SQlite
considering compatibility issues
conn = sqlite3.connect('my_data1.db')

# Create a cursor object
cur = conn.cursor()

# Execute the SQL query to get 5 rows where launch sites begin
with KSC string

cur.execute('SELECT * FROM SPACEXTABLE WHERE
 UPPER("LAUNCH_SITE") LIKE "KSC%" LIMIT 5')
# Fetch all results
results = cur.fetchall()
# Print the results
for row in results:
   print(row)
# Close the connection
conn.close()
```

Upper is used to check for capital letters; like allows us to consider any trail or space. Limit 5 sets a limit to the number of rows to show. The first 5 rows show that these were successes both for drone ships and ground pad

Results of the query:

```
('2017-02-19', '14:39:00', 'F9 FT B1031.1', 'KSC LC-39A', 'SpaceX CRS-10', 2490, 'LE
O (ISS)', 'NASA (CRS)', 'Success', 'Success (ground pad)')
('2017-03-16', '6:00:00', 'F9 FT B1030', 'KSC LC-39A', 'EchoStar 23', 5600, 'GTO', 'E
choStar', 'Success', 'No attempt')
('2017-03-30', '22:27:00', 'F9 FT  B1021.2', 'KSC LC-39A', 'SES-10', 5300, 'GTO', 'SES
', 'Success', 'Success (drone ship)')
('2017-05-01', '11:15:00', 'F9 FT B1032.1', 'KSC LC-39A', 'NROL-76', 5300, 'LEO', 'N
RO', 'Success', 'Success (ground pad)')
('2017-05-15', '23:21:00', 'F9 FT B1034', 'KSC LC-39A', 'Inmarsat-5 F4', 6070, 'GTO',
'Inmarsat', 'Success', 'No attempt')
```

Total Payload Mass

```
# Changing the code to allow for connection via SQlite considering compatibility issues
conn = sqlite3.connect('my_data1.db')

# Create a cursor object
cur = conn.cursor()

# Execute the SQL query to get total of the payload mass when payload has 'CRS' in it - NASA launches

cur.execute('SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE UPPER("Payload") LIKE "%CRS%"')

# Fetch all results
results = cur.fetchall()

# Print the results
for row in results:
   print(row)

# Close the connection
conn.close()
```

Upper is used to check for capital letters; like allows us to consider any trail or space. Sum() allows us to total the payload mass with a filter criteria using the 'Where' clause. The total payload mass from NASA launches is noted.

Result of the query:

```
(111268,)
```

Average Payload Mass by F9 v1.1

```
# Changing the code to allow for connection via SQlite considering compatibility
issues
conn = sqlite3.connect('my_data1.db')

# Create a cursor object
cur = conn.cursor()

# Execute the SQL query to get average of the payload mass when booster version
is F9 v1.1

cur.execute('SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE
 UPPER("Booster_Version") LIKE "F9 v1.1"')

# Fetch all results
results = cur.fetchall()

# Print the results
for row in results:
   print(row)

# Close the connection
conn.close()
```

Upper is used to check for capital letters; like allows us to consider any trail or space. Avg() allows us to total the payload mass with a filter criteria using the 'Where' clause. The average payload is around 3000 (exact number is 2928.4) for the booster version of F9 v1.1

```
Result of the query:

(2928.4,)
```

First Successful Drone Landing Date

- Find the dates of the first successful landing outcome on drone ship.Present your query result with a short explanation *# Changing the code to allow for connection via SQlite considering compatibility issues*
- conn = sqlite3.connect('my_data1.db')

- *# Create a cursor object*
- cur = conn.cursor()
- *# Execute the SQL query to get date when successful landing outcome in drone ship was achieved (landing outcome is Success (drone ship))*
- cur.execute("SELECT MIN(Date) FROM SPACEXTABLE WHERE Landing_Outcome ='Success (drone ship)'")

- *# Fetch all results*
- results = cur.fetchall()

- *# Print the results*
- **for** row **in** results:
-     print(row)

Min() allows us to take the earliest date for the successful launch with a filter criteria using the ‘Where’ clause. The earliest date of the successful launch was 8 April 2016.

Result of the query:
('2016-04-08',)

Successful Ground Landing with Payload between 4000 and 6000

- *# Changing the code to allow for connection via SQlite considering compatibility issues*
- conn = sqlite3.connect('my_data1.db')
- *# Create a cursor object*
- cur = conn.cursor()
- *# Execute the SQL query to get booster version when successful landing outcome in ground pad and payload mass between 4000 to 6000*
- cur.execute("SELECT BOOSTER_VERSION FROM SPACEXTABLE WHERE Landing_Outcome = 'Success (ground pad)' AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000")
- *# Fetch all results*
- results = cur.fetchall()
- *# Print the results*
- **for** row **in** results:
-     print(row)

and() allows us to take a range of payload (between) with a filter criteria using the 'Where' clause. There are three booster versions that fit the criteria

Result of the query:

```
('F9 FT B1032.1',)
('F9 B4 B1040.1',)
('F9 B4 B1043.1',)
```

Total Number of Successful and Failure Mission Outcomes

```
# Changing the code to allow for connection via SQlite considering compatibility
issues

conn = sqlite3.connect('my_data1.db')

# Create a cursor object

cur = conn.cursor()
# Execute the SQL query to get number of successful and failure mission outcomes

cur.execute("SELECT CASE WHEN LANDING_OUTCOME LIKE '%Success%' THEN
 'Success' WHEN LANDING_OUTCOME LIKE '%Failure%' THEN 'Failure'END AS
 General_Outcome,COUNT(*) AS Outcome_Count FROM SPACEXTABLE WHERE
 LANDING_OUTCOME LIKE '%Success%' OR LANDING_OUTCOME LIKE '%Failure%
' GROUP BY General_Outcome")


# Fetch all results
results = cur.fetchall()

# Print the results
for row in results:
   print(row)

# Close the connection
conn.close()
```

Group by() allows us to group the numbers based on landing outcome and a case when works to assign numbers based on the outcome. There are three booster versions that fit the criteria. 10 failures and 61 successes are noted.

Result of the query:

```
('Failure', 10)
('Success', 61)
```

Boosters Carried Maximum Payload

```
# Changing the code to allow for connection via SQlite considering compatibility
issues
conn = sqlite3.connect('my_data1.db')

# Create a cursor object
cur = conn.cursor()
# Execute the SQL query to get booster version payload mass is maximum
cur.execute("SELECT BOOSTER_VERSION FROM SPACEXTABLE WHERE
PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM
SPACEXTABLE)")

# Fetch all results
results = cur.fetchall()

# Print the results
for row in results:
    print(row)

# Close the connection
conn.close()
```

Where allows for a filter criteria and max() lets us take the maximum of the payload . F9 B5 booster versions seem to have the maximum payloads.

Result of the query:

```
('F9 B5 B1048.4',)
('F9 B5 B1049.4',)
('F9 B5 B1051.3',)
('F9 B5 B1056.4',)
('F9 B5 B1048.5',)
('F9 B5 B1051.4',)
('F9 B5 B1049.5',)
('F9 B5 B1060.2 ',)
('F9 B5 B1058.3 ',)
('F9 B5 B1051.6',)
('F9 B5 B1060.3',)
('F9 B5 B1049.7 ',)
```

2017 Launch Records

```
# Changing the code to allow for connection via SQlite considering
compatibility issues
conn = sqlite3.connect('my_data1.db')

# Create a cursor object
cur = conn.cursor()
# Execute the SQL query to get month names and other variables as
needed
cur.execute("SELECT CASE WHEN substr(Date, 6, 2) = '01' THEN
'January' WHEN substr(Date, 6, 2) = '02' THEN 'February' WHEN
substr(Date, 6, 2) = '03' THEN 'March' WHEN substr(Date, 6, 2) = '04'
THEN 'April' WHEN substr(Date, 6, 2) = '05' THEN 'May' WHEN
substr(Date, 6, 2) = '06' THEN 'June' WHEN substr(Date, 6, 2) = '07'
THEN 'July' WHEN substr(Date, 6, 2) = '08' THEN 'August' WHEN
substr(Date, 6, 2) = '09' THEN 'September' WHEN substr(Date, 6, 2) =
'10' THEN 'October' WHEN substr(Date, 6, 2) = '11' THEN 'November'
WHEN substr(Date, 6, 2) = '12' THEN 'December' END AS
Month_Name, LANDING_OUTCOME, BOOSTER_VERSION,
LAUNCH_SITE FROM SPACEXTABLE WHERE substr(Date, 0, 5) = '2017'
AND LANDING_OUTCOME LIKE 'Success (ground pad)'")
# Fetch all results
results = cur.fetchall()

# Print the results
for row in results:
   print(row)

# Close the connection
conn.close()
```

A case when works to check for conditions to meet and the date field is split to consider months. The where clause is used to check for the year needed and the success outcome is checked for ground landings. There are 6 records that meet this criteria. Most of then were launched from KSC LC-39 A, one is from CCAF S SLC-40.

Result of the query:

```
('February', 'Success (ground pad)', 'F9 FT B1031.1', 'KSC
LC-39A')
('May', 'Success (ground pad)', 'F9 FT B1032.1', 'KSC LC-3
9A')
('June', 'Success (ground pad)', 'F9 FT B1035.1', 'KSC LC-3
9A')
('August', 'Success (ground pad)', 'F9 B4 B1039.1', 'KSC LC
-39A')
('September', 'Success (ground pad)', 'F9 B4 B1040.1', 'KS
C LC-39A')
('December', 'Success (ground pad)', 'F9 FT  B1035.2', 'CC
AFS SLC-40')
```

Rank Landing Outcomes Between 2010-06-04 and 2017-03-20

```
# Changing the code to allow for connection via SQlite considering compatibility issues
conn = sqlite3.connect('my_data1.db')

# Create a cursor object
cur = conn.cursor()
# Execute the SQL query to consider landing outcomes count between 2010-06-04 and 2017-03-20 in descending order
cur.execute("SELECT LANDING_OUTCOME, COUNT(*) AS Outcome_Count FROM SPACEXTABLE WHERE Date BETWEEN '2010-06-04' AND '2017-03-20' GROUP BY LANDING_OUTCOME ORDER BY Outcome_Count DESC")
# Fetch all results
results = cur.fetchall()

# Print the results
for row in results:
   print(row)

# Close the connection
conn.close()
```

The where clause is used to check for the dates needed and the success outcome is checked for by grouping by the landing outcome. Order by descending allows us to first list the success launches (as they have a value of 1) and then failures.   Drone ship landing has a 50% success rate. 2 failures noted through parachute landing. No attempts saw 10 cases.

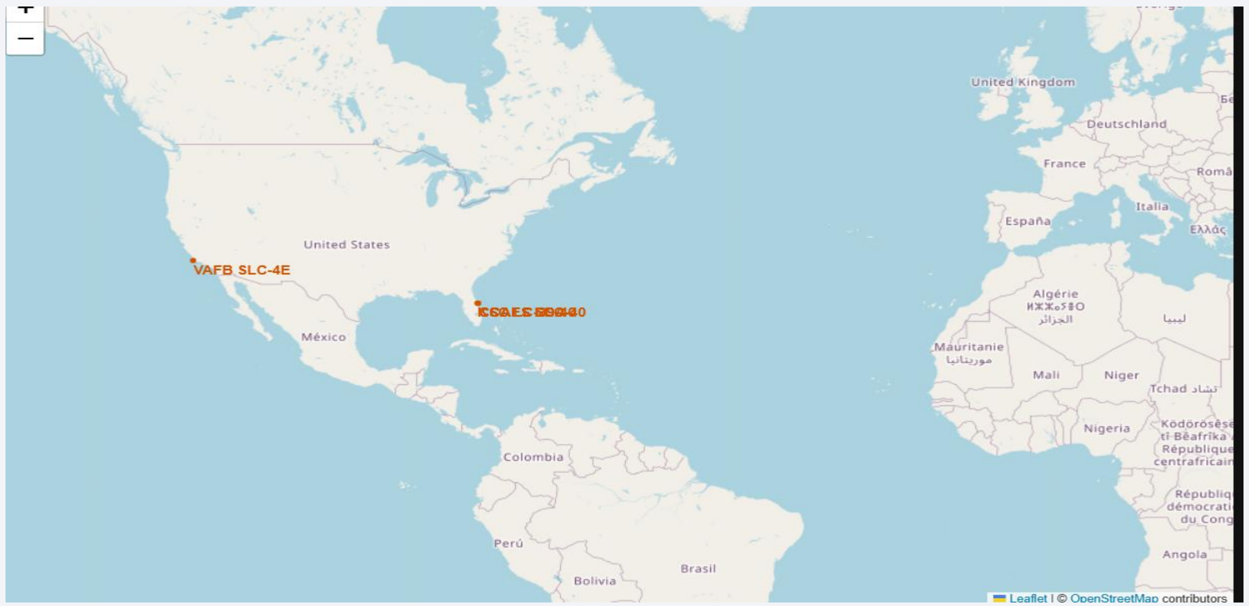
Result of the query:

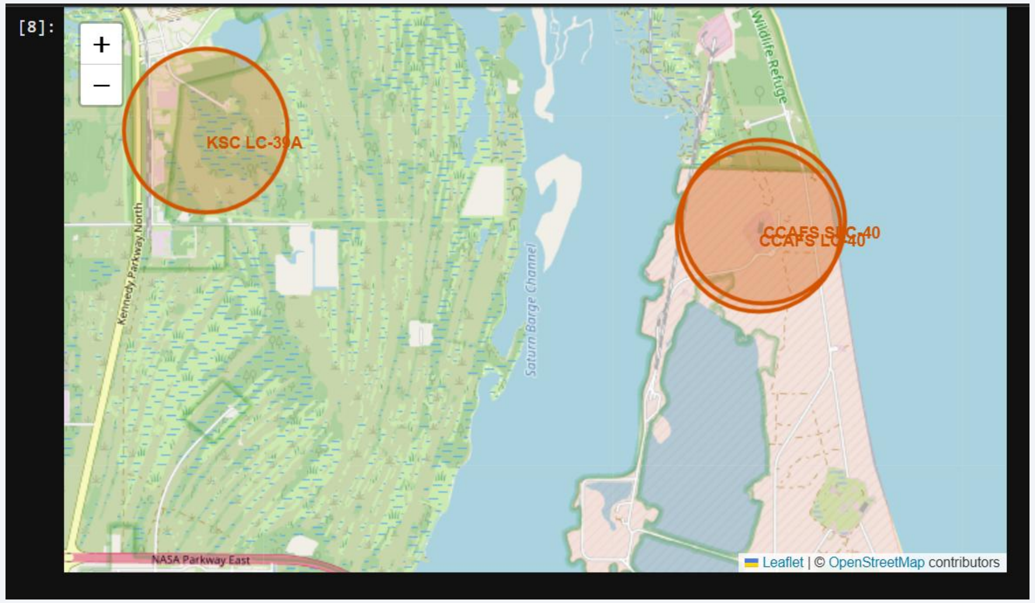
```
('No attempt', 10)
('Success (drone ship)', 5)
('Failure (drone ship)', 5)
('Success (ground pad)', 3)
('Controlled (ocean)', 3)
('Uncontrolled (ocean)', 2)
('Failure (parachute)', 2)
('Precluded (drone ship)', 1)
```

Section 3

Launch Sites Proximities Analysis

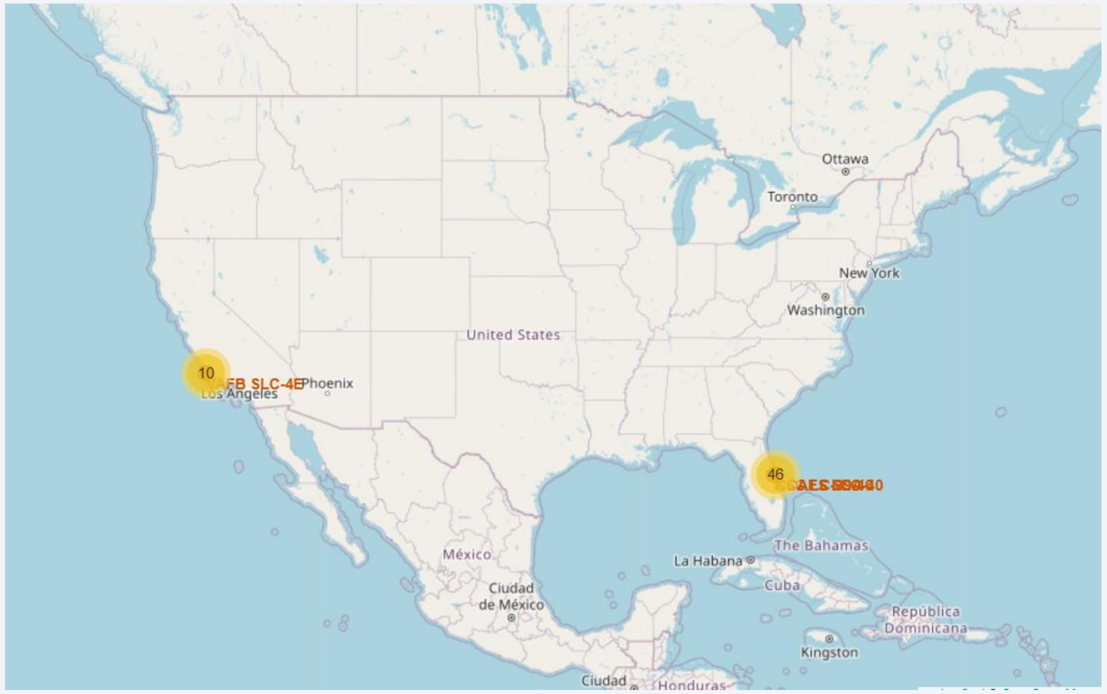
Proximity of launch sites to coastlines

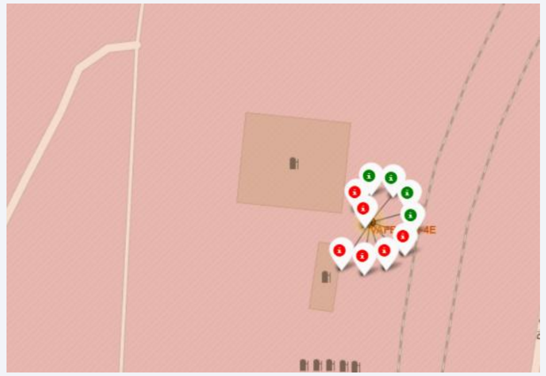


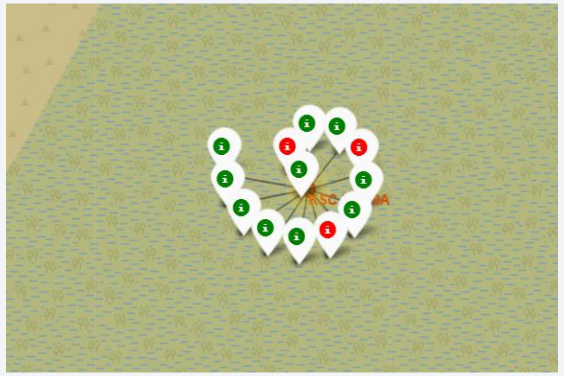


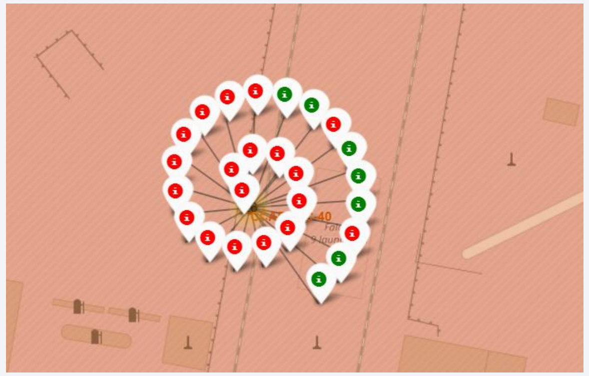
Launch sites are all in the US (as SpaceX is a US based company) and are all in close proximity to coastlines. This allows for a safe launch of the rockets.

Launch Outcomes by site







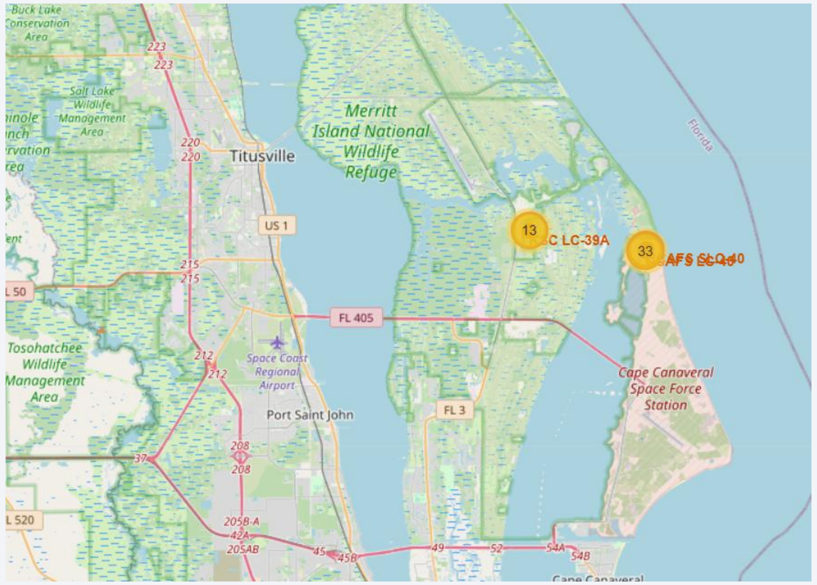


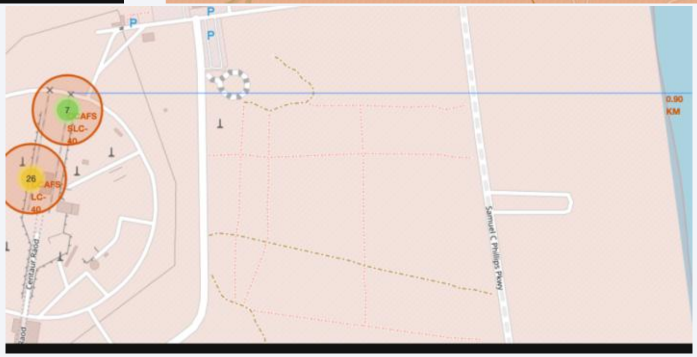
The first map shows the clusters marked for each site; the next few show outcomes marked against each site through the use of markers. KSC LC-39A has the maximum success rate noted. The maximum number of launches are from CCAFS LC-40

Safety vs Logistics TradeOff for Launch Site proximities

7
CCAFS SLC-40
26
CCAFS LC-40
Centaur Road

CCAFS SLC-40
33
CCAFS LC-40
Samuel C Phillips Parkway

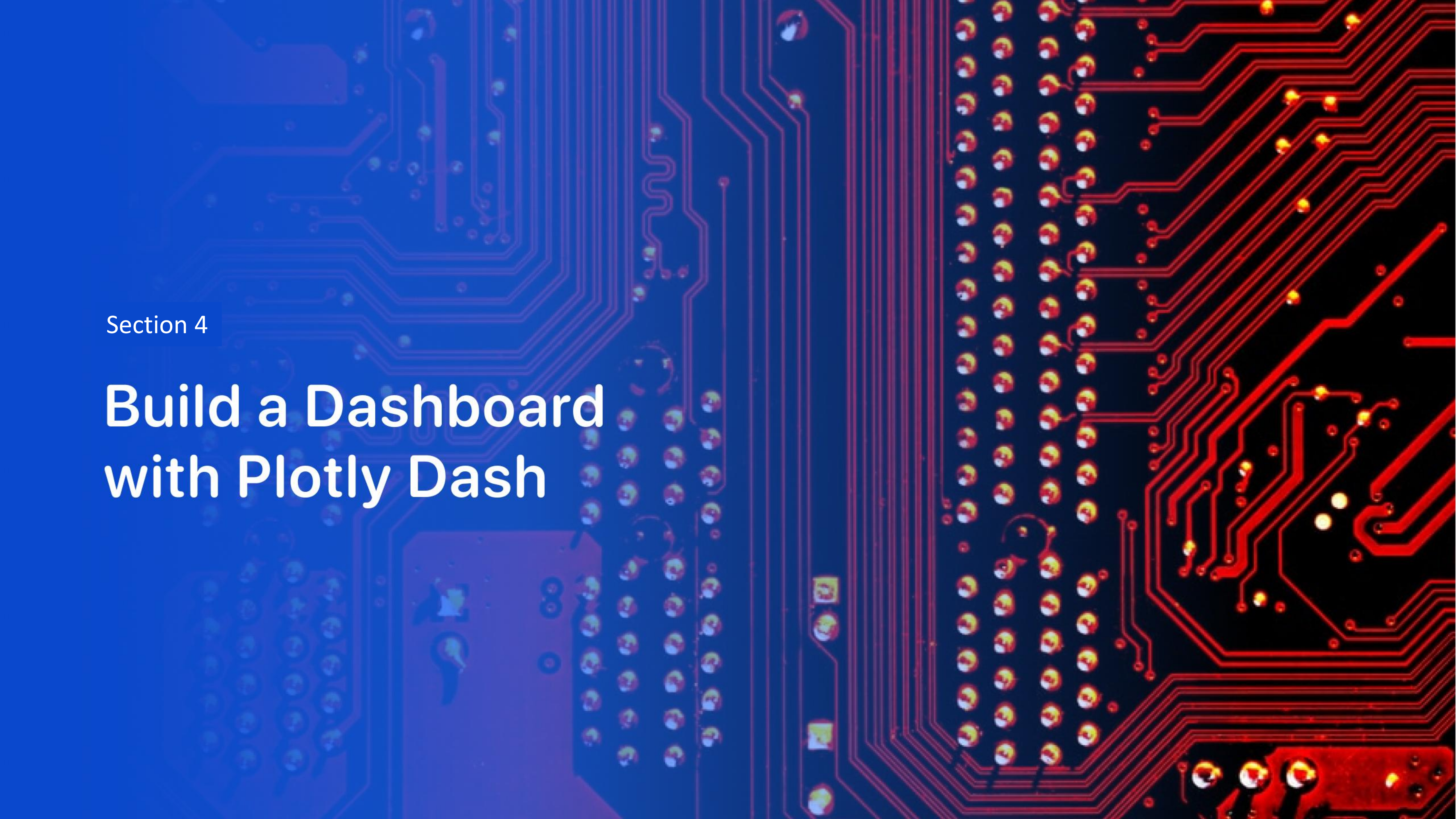

Titusville
Merritt Island National Wildlife Refuge
13
KSC LC-39A
33
CCAFS SLC-40
Tosohatchee Wildlife Management Area
Space Coast Regional Airport
Port Saint John
Cape Canaveral Space Force Station
US 1
FL 405
FL 3



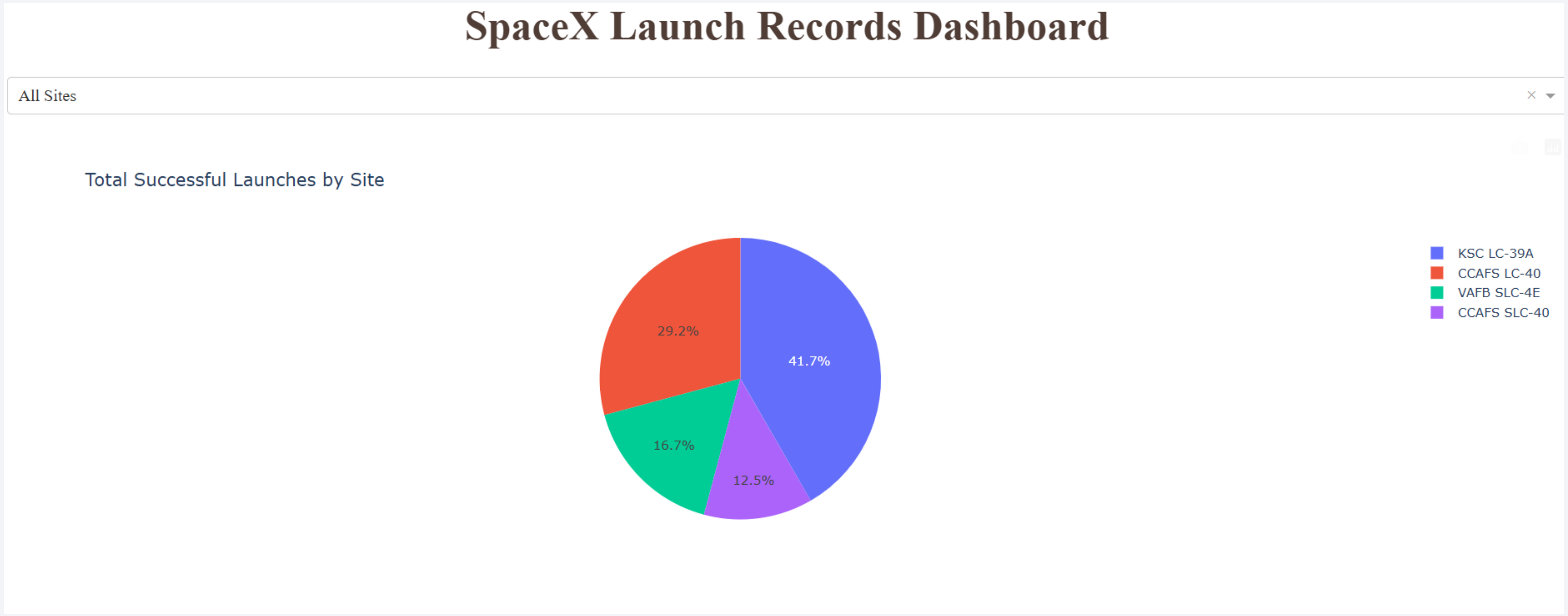
Launch sites are generally in reserved areas or far off areas to ensure safety to people and property in case of issues/glitches – however they should still be relatively close to railways, roads and the coastline. It should also not be too inaccessible from cities as people working at these sites will need to be able to move around.

Section 4

Build a Dashboard with Plotly Dash



Launch Outcomes by Launch Sites



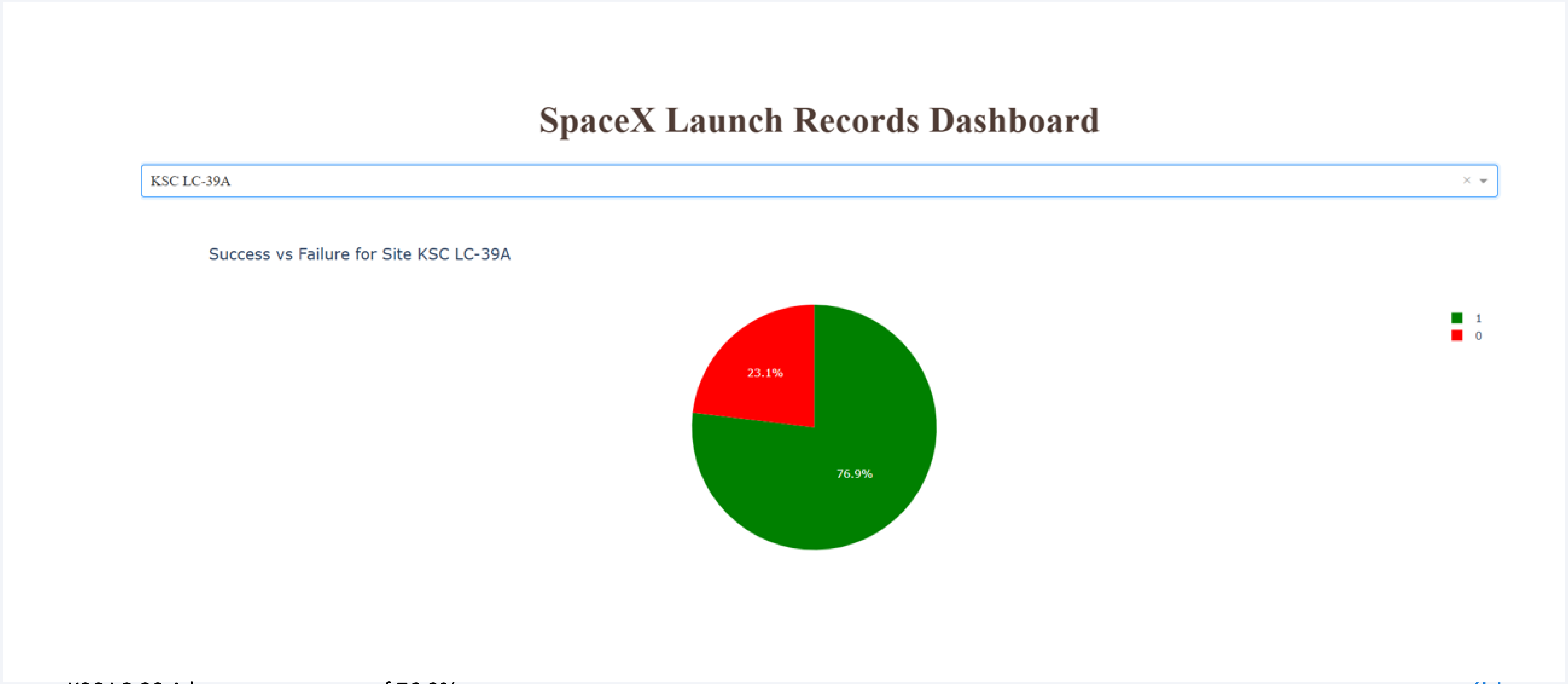
SpaceX Launch Records Dashboard

All Sites

Total Successful Launches by Site

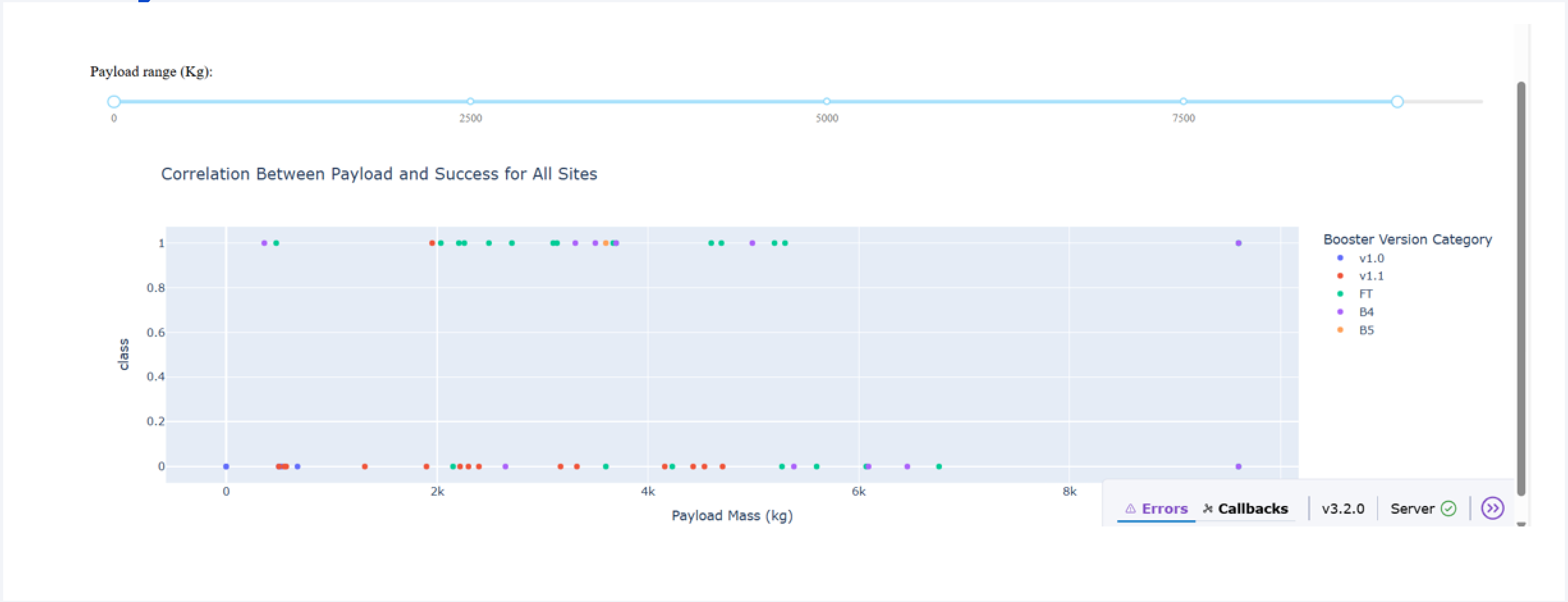
KSC LC-39 A has the higher success rates noted while the least is seen for CCAFS SLC-40.

Launch site with highest launch success ratio – KSC LC-39A



KSC LC-39 A has a success rate of 76.9%.

Payload vs. Launch Outcome

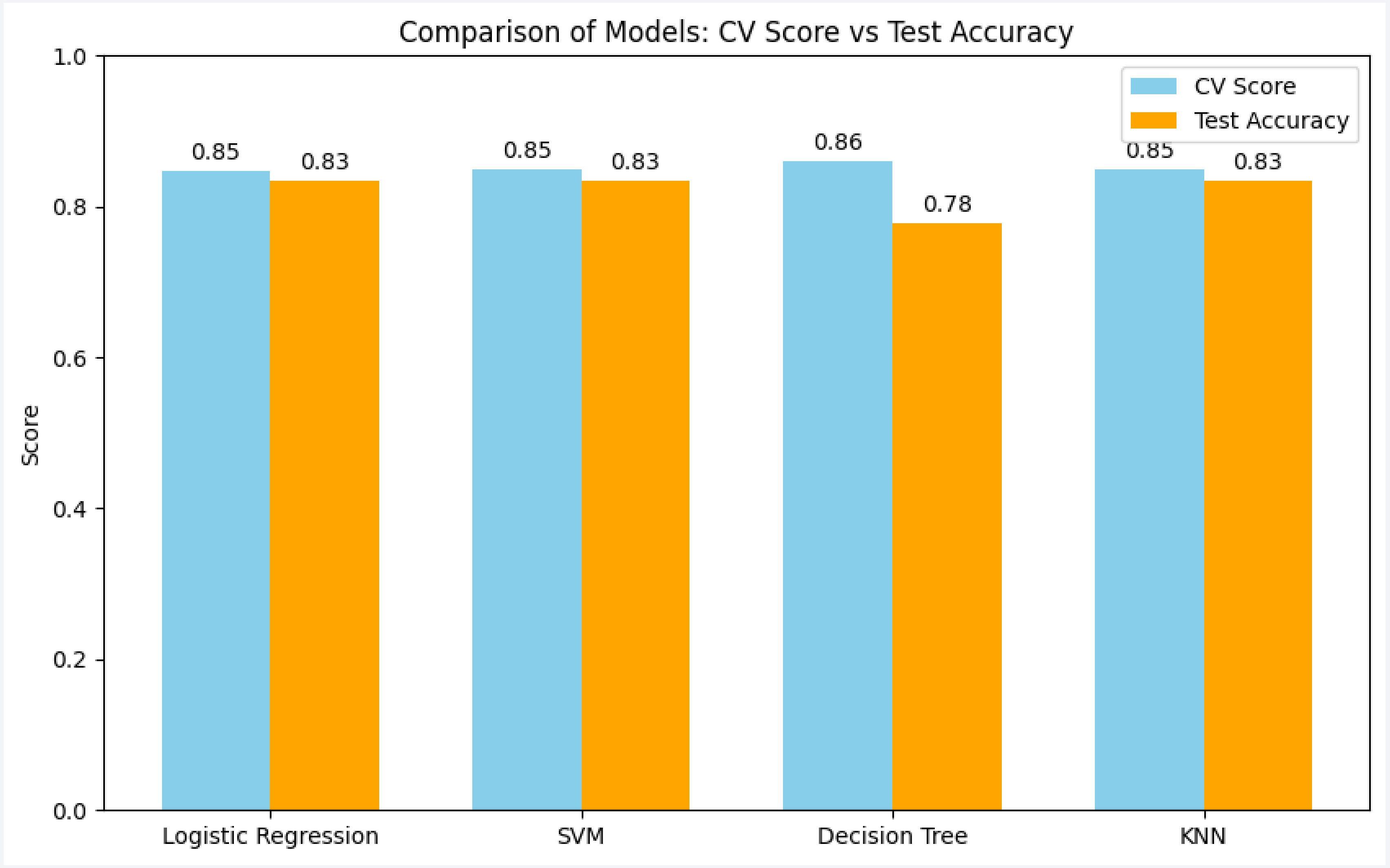


Payloads between 2000 – 5000 have the highest success rates. We see more failures for lower and higher payloads, however, the launches with booster v1.1 version show more tendency to fail across most payload ranges. Higher payloads for B4 version show some success

Section 5

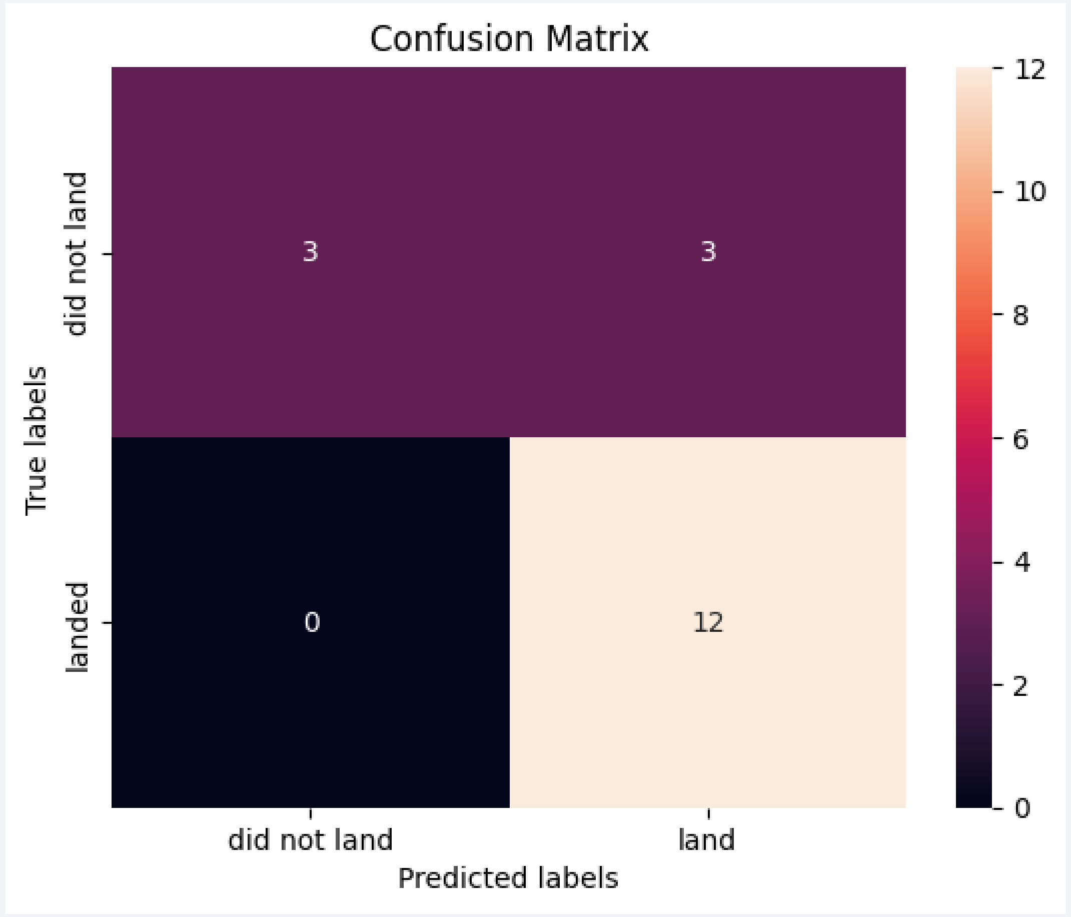
Predictive Analysis (Classification)

Classification Accuracy



Decision tree models show overfitting to the training data as accuracy drops for the test data. The other three models all perform with similar accuracy.

Confusion Matrix



Confusion matrix for three of the models – logistic regression, SVM and KNN are similar to each other. We have 12 true positives and 3 false positives for landing.

Conclusions

- Decision tree model while performing the best for training data does not perform as well for the test data. It is hence considered a problem of overfitting to the data it has been trained on (a classic issue with decision tree classification techniques noted)
- All other models perform similar to each other with approx. 83% accuracy
- A problem of false positives is noted from the confusion matrix for models other than the decision tree.
- Success rates through time increase as experience allows SpaceX to finetune their process.
- We can predict launch outcome to an accuracy level of 83% using the parameters of launch site, payload mass, number of flights (that allows experience to be built in) as well as booster versions that will help estimate costs .

Appendix

All codes and data are available in https://github.com/rajalakshmi130880/IBM-DS0720EN-Data-Science-and-Machine-Learning-Capstone-Project.git

Thank you!